Editing Files 10

Editing and creating files are integral parts of programming. This chapter describes features of the built-in THINK editor. The editor uses standard Macintosh editing techniques, so you're familiar with its basic operation. Although you can use it to edit any text file, the editor has some features that make editing your source and header files easier.

Contents

Creating and Opening Files 165
- Creating a new file 165
- Opening a text file 165
- Opening text files automatically 166
- Opening a source file 166
- Opening header files 166
- Opening the current selection 167

Editing a File 168
- Typing text 168
- Undoing changes to a file 168
- Selecting lines 168
- Indenting 169
- Shifting blocks right and left 169
- Balancing parentheses, brackets, and braces 169
- Changing font and tab settings 170

Moving Around a File 170
- Using the arrow and function keys 171
- Scrolling to the insertion point 172
- Moving to a specific line 172

Looking Up Definitions and Documentation 173
- Looking up function, class, and global variable definitions . 173
- Looking up method definitions 173
- Looking up ANSI and Macintosh Toolbox documentation . 174

Printing Files and Windows 174

Selecting and Arranging Windows 175

Closing and Saving Files 176
Closing a file 176
Saving a file 176
Saving a file with a different name 176
Saving and closing all open files 177
Saving files automatically 177

Executing MPW Commands 177

Using Markers 179
Adding markers 179
Going to a marker 179
Deleting a marker 181

Using Your Own Editor 181
Using a THINK-compatible editor 181
Using another editor 182

Searching and Replacing 182
Finding a string 183
Search options 184
Replacing a string 185
Setting things up for searching later 185
Finding nonprinting characters 186
Searching through multiple files 186
Disabling multi-file search 190

Searching for a Pattern (Grep) 190
Patterns 190
Simple patterns 190
Complex Patterns 191
Sub-patterns 192
Constraining patterns 193
Replacing with Grep 193
Grep Examples 194

Creating and Opening Files

To learn how to open a project, see "Opening an Existing Project" on page 133.

To create or open a file, you must first have a project window open. You can open as many files as the memory in your Macintosh allows, and each file appears in its own edit window. Although you usually use the THINK editor to create and open source or header files, it can also open any text file.

Creating a new file

To create a new file, choose **New** from the **File** menu. An untitled edit window appears, ready for you to start typing into it.

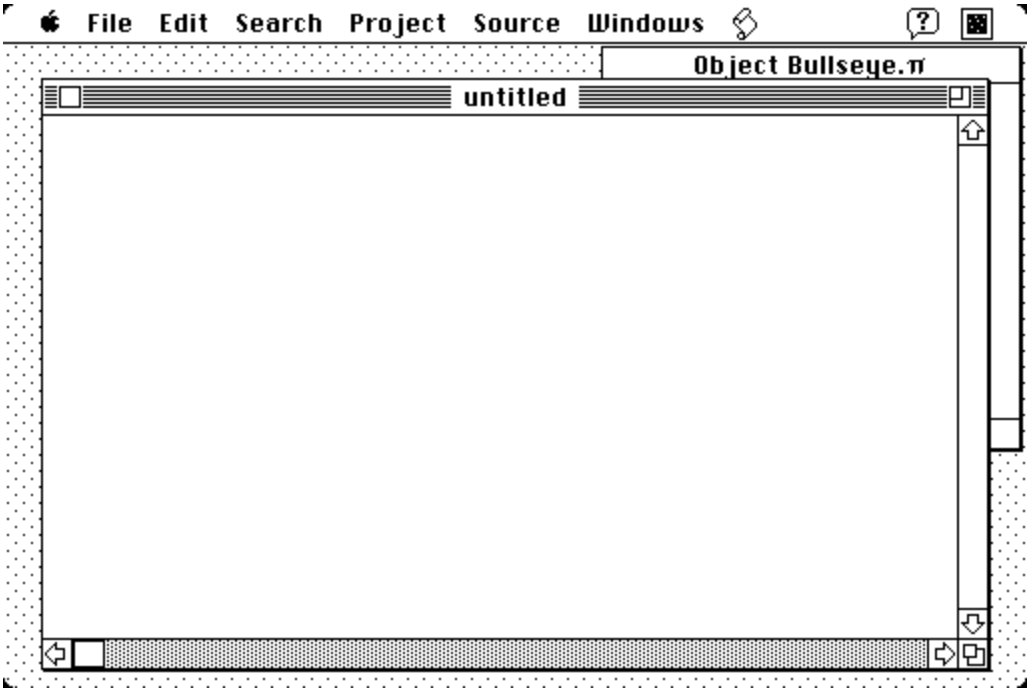

Figure 10-1 New editing window

Opening a text file

The **Open** command in the **File** menu opens any text file. A standard file dialog box displays the names of all text files in the current folder, even if they weren't created with the THINK editor. The file you open appears in its own edit window.

Opening text files automatically

The THINK Project Manager can open some text files automatically whenever it opens a project. If you select the Reopen files option in the Editor page of the **THINK Project Manager options** dialog, the THINK editor remembers which of the project's files were open when you closed the project and reopens those files when you next open the project. Files that are not in the project aren't reopened.

Opening a source file

To open a source file that's already in your project window, double-click its name in the project window. If the file is already open, double-clicking its name brings the file's edit window to the front.

You can select a file by typing the first few characters of its name when the project window is the active window. You can then open the file by pressing Return or Enter. Since the files in the project window are in alphabetical order, the selected file is the first file in the project that matches the characters typed so far. If what you type doesn't match any name in the project window, any selected file is deselected. If it matches more than one file name, you can use the Tab key to cycle among all the names that match. The Up and Down Arrow cursor keys also change the selection in the project window.

Note

If your keyboard doesn't have arrow keys, use the Backspace key. In the project window, use Backspace as Up Arrow, and Shift-Backspace as Down Arrow.

Opening header files

Sometimes when you're working on a source file, you want to look at the header files associated with it. Of course, you can use the **Open** command from the **File** menu to open header files, but the THINK editor gives you a way to open these files more more quickly.

Holding down the Command key while clicking a title bar brings up a menu of markers, described in "Using Markers" later in this chapter.

To get a pop-up menu of all the files included in a source file, as shown in Figure 10-2, hold down the Option key as you click the title bar of its window. To open a header file in its own edit window,

choose its name. If the file is already open, the THINK editor brings its window to the front.

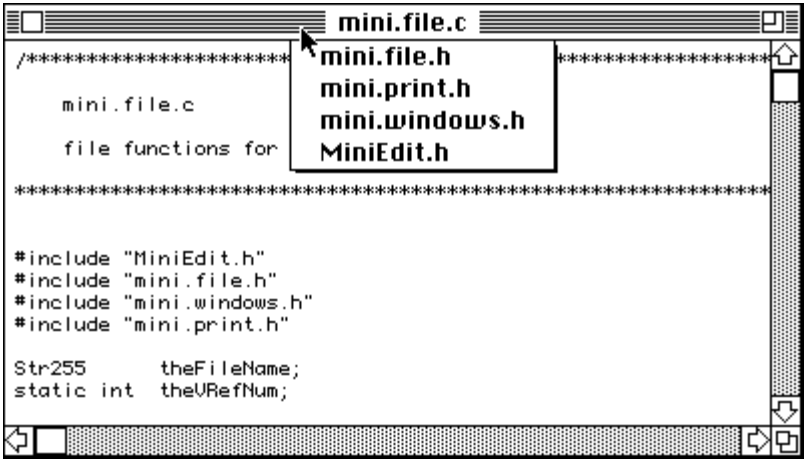

Figure 10-2 The list of a file's header files

Holding down the Option key as you click the title bar of the project window brings up a pop-up menu containing names of all the header files used in the project.

Note

The THINK editor builds the pop-up menus only for compiled source files. Any header files that are part of a precompiled header don't appear in the pop-up menus.

To open a header file that you've added to a source file since the last compilation, or to open a header file for a file you haven't yet compiled, use the **Open Selection** command described below.

Opening the current selection

The **Open Selection** command in the **File** menu lets you open a header file by selecting its name in the current source file (double-clicking the name). You don't have to select the `.h` extension. The selection is extended automatically to include it, as long as you've selected the first part of the file name.

Open Selection can deal with path names. If the character following the selection is `.` (period) or `:` (colon), the selection is

To learn about the project and THINK trees, see Chapter 9, "Organizing Files and Folders."

extended to the end of the next word, and this process is repeated. A partial path name is searched for as though it appeared in an `#include "..."` statement in the file being edited. (**Open Selection** is not smart enough to look for angle brackets.) If the edit window is untitled, only the project and THINK trees are searched.

Editing a File

The editor uses all the standard Macintosh editing techniques as well as some techniques designed for editing C programs. Double-clicking a word selects the entire word. Triple-clicking a line selects the entire line. **Cut**, **Copy**, **Paste**, and **Undo** all work in the same way as in other Macintosh applications.

Typing text

The THINK editor does not have the word-wrap feature you might be used to in other editors. If you type past the right edge of the window, the window automatically scrolls horizontally so you can still see the insertion point.

Undoing changes to a file

If you make an unintentional change to your file, use the **Undo** command from the **Edit** menu. **Undo** remembers only the last thing you did. You can also use **Undo** to redo what you undid. The wording of the **Undo** command always reflects what it will undo. For example, if you cut a range of lines, the **Undo** command reads **Undo Cut**. If you choose **Undo** again, the command now says **Redo Cut**. You can undo the most recent **Replace** (see "Searching and Replacing" later in this chapter), but you can't undo a **Replace All** command.

If you've made numerous changes to your file and you want to undo all of them, or if you want to undo a **Replace All** command use the **Revert** command in the **File** menu. **Revert** discards all the changes you made since you last saved (or opened) the file.

Selecting lines

To select a line, triple-click anywhere in the line. To select a range of lines, drag the mouse after you triple-click.

Indenting

When you press Return to start a new line, the editor indents the new line with the same number of leading tabs and spaces as the previous line has.

To change the indentation, backspace over it. To keep the editor from auto-indenting a line, hold down the Option key when you press Return.

Shifting blocks right and left

To change the indentation level for a range of lines, use the **Shift Left** and **Shift Right** commands from the **Edit** menu. **Shift Left** deletes the leading tab from each selected line. **Shift Right** inserts a tab at the beginning of each selected line. Both **Shift Left** and **Shift Right** extend the current selection to include entire lines.

Warning

Do not type anything while doing **Shift Left** or **Shift Right** on a selected region. Typing replaces the selected text with whatever you typed.

Balancing parentheses, brackets, and braces

The **Balance** command on the **Edit** menu extends the current selection in both directions until it encloses the smallest block of text enclosed in parentheses `( )`, brackets `[ ]`, or braces `{ }`. Successive invocations select larger sequences of text.

Note

The **Balance** command looks for any parentheses, brackets, and braces. It does not ignore text in strings or comments.

Balance scrolls automatically to the beginning of the text it selects. To scroll to the end, press Enter. To scroll back to the beginning, press Enter again.

You can use **Balance** to verify that all the parentheses, brackets, and braces in your file are balanced properly. Go to the beginning of your file and search for the first left brace ({). Use **Balance** and **Find Again** repeatedly until you get to the end of your file.

Changing font and tab settings

When you open a new edit window with the **New** command, the font is 9-point Monaco, and there's a tab stop every four spaces. You can change these settings with the **Set Tabs & Font** command on the **Edit** menu. When you choose this command, you see the dialog in Figure 10-3.

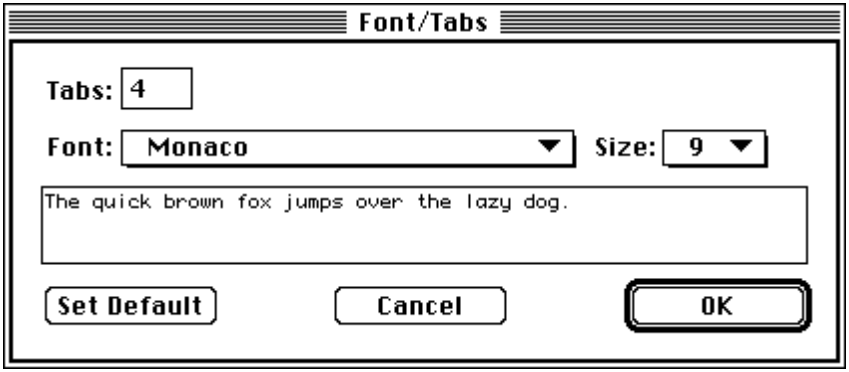

Figure 10-3 The Set Tabs & Font dialog

Type a number to set the number of spaces per tab. To change the font, click the font name and select a font from the pop-up menu that appears. To change the font size, click the size and choose a number from the pop-up menu is displayed.

Click OK to set the font and tabs for the current file. Click Set Default to set the default font and tab size for all files that don't have font and tab information. Click Cancel if you decide not to make a change.

If you are using a proportionally spaced font such as New York or Geneva, the THINK editor uses the width of the nonbreaking space (Option-Space) to figure out the width of a tab.

When you change the font or tab settings for an individual file, the editor adds an `MPSR` resource to your text files to record the new settings. Other text editors use this resource as well.

Moving Around a File

The THINK editor gives you several ways to move around your files quickly. If you don't want to take your hands off the keyboard, you don't ever need to click a scroll bar.

Using the arrow and function keys

The arrow keys move the insertion point up, down, left, or right. At the end of a line, the left (or right) arrow wraps around to the end of the previous line, or beginning of the next line. Holding the Command key and pressing any arrow key moves the insertion point as far up, down, left, or right as possible. Shift-Arrows and Shift-Command-Arrows extend the current selection.

Table 10-1 summarizes how the arrow and editing keys move the insertion point.

Press this…	To move the insertion point…
Up Arrow	Up one line
Down Arrow	Down one line
Left Arrow	Left one character
Right Arrow	Right one character
Option-Left Arrow	Left one word
Option-Right Arrow	Right one word
Command-Up Arrow	To the beginning of the file
Command-Down Arrow	To the end of the file
Command-Left Arrow	To the beginning of the line
Command-Right Arrow	To the end of the line

Table 10-1 Using the arrow keys in the editor

Note

If you're using a Macintosh Plus keyboard, the editor treats the +, *, /, and = keys on the numeric keypad like Shift-Arrow keys. For example, the + key on the keypad is treated as a Shift-Left Arrow. On other keyboards, the +, *, /, and = keys work correctly.

If you have an Apple Extended Keyboard, you can use the keys above the arrow keys. Table 10-2 describes what these keys do.

Press this…	To do this…
Forward Delete (⌦)	Delete the next character
Home	Scroll to the beginning of the file
End	Scroll to the end of the file
Page Up	Scroll to the previous screen
Page Down	Scroll to the next screen

Table 10-2 Using the editing keys in the editor

Press this…	To do this…
Option-Page Up	Scroll to the left
Option-Page Down	Scroll to the right
Option-Home	Scroll all the way to the left
Option-End	Scroll all the way to the right

Table 10-2 Using the editing keys in the editor *(Continued)*

Note

The Home, End, Page Up, and Page Down keys do not move the insertion point. They just scroll the file.

Scrolling to the insertion point

The THINK editor lets you examine an area of your file, then jump back instantly to where you were. To see the current insertion point or the start of the current selection, press Enter. Because scrolling in any direction does not affect the insertion point or the current selection, pressing Enter takes you back to your previous position in the file.

If the start of the selection is already visible, pressing Enter makes the end of the selection visible, so you can toggle between the two ends of the current selection. This shortcut is particularly useful after using the **Balance** command (see "Balancing parentheses, brackets, and braces" earlier in this chapter).

Moving to a specific line

The **Go To Line** command in the **Search** menu lets you move to a specific line in your file. Lines are numbered consecutively, with the first line in the file being number 1.

When you choose **Go To Line**, you see the dialog in Figure 10-4.

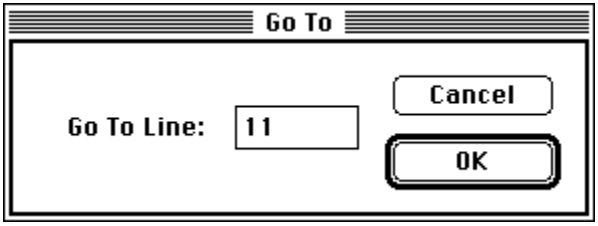

Figure 10-4 The Go To Line dialog

When the dialog is displayed, the line number specified is the line that contains the insertion point. To go to a particular line, type its number and click OK. If you change your mind, click Cancel. The editor moves the insertion point to the beginning of the line you entered.

Looking Up Definitions and Documentation

The THINK editor lets you look up definitions for functions, classes, and global variables that you define in your project. In addition, if you have THINK Reference 2.0 or later, you can look up documentation on any Macintosh Toolbox function, type, or variable.

Before you look up definitions with these shortcuts, you must compile your project. You can look up documentation in THINK Reference without compiling your project.

Looking up function, class, and global variable definitions

To look up the definition of a function, class, or global variable that's defined in your project, hold down the Option key and double-click the name of the symbol. The THINK editor opens the file that defines the function, class, or global variable, and moves to the first occurrence of the symbol.

Note

If the THINK editor doesn't bring you to the definition of the symbol, use the **Find Again** command until you locate it.

Looking up method definitions

Option-double-click the method name. If more than one class defines a method with that name, the THINK Project Manager displays the Class Browser and highlights the classes that define a method for that message. Double-click the class you want. The THINK editor opens the file that defines it and moves to the definition. For more information on the Class Browser, see Chapter 14, "Browsing Classes."

To look up the definition of a class constructor, hold down both the Option and Command keys as you double-click the constructor name.

Looking up ANSI and Macintosh Toolbox documentation

The THINK Project Manager comes with a version of the THINK Reference that documents the ANSI library. If you have Symantec C++, it also comes with a database that documents IOStreams and Complex libraries. If you have the version of THINK Reference 2.0 that's available in stores, you have more databases that let you look up documentation on the functions, types, and variables documented in *Inside Macintosh I–VI*. THINK Reference is available wherever you bought THINK C or Symantec C++.

If you're running System 7, you can look up information in THINK Reference from the THINK editor. If you select a function, type, or variable name described in one of the libraries and you choose **Find in THINK Reference** from the **Search** menu, the THINK editor launches THINK Reference, which opens to the page that describes the symbol. If THINK Reference is already running and you Option-double-click a symbol, the THINK editor switches to THINK Reference, which opens to the page that describes the symbol. To return to the THINK Project Manager, click the Iconize button in the THINK Reference window.

Note

THINK Reference 1.0 does not support this feature. You must have THINK Reference 2.0 or later.

If you don't want the THINK editor to switch to THINK Reference when you want to look up the definition of a symbol, hold down both the Control and Option keys when you double-click the symbol.

If you followed the installation instructions according to Chapter 2, "Installation," this feature works without any additional set up. Otherwise, make sure there's an alias to THINK Reference 2.0 in the `Tools` folder in the THINK Project Manager folder. You must name the alias `THINK Reference`.

Printing Files and Windows

To print the contents of the frontmost window, choose **Print** from the **File** menu. You see the standard print dialog for your printer. You can print the contents of an edit window, project window, Search Results window, Compile Errors window, and Link Errors window.

The **File** menu also contains a standard **Page Setup** command that lets you set the page size and other options before you print.

Selecting and Arranging Windows

When you have several files on the screen, it can be difficult to find a particular one. The THINK Project Manager has several commands to help you.

The bottom of the **Windows** menu lists the editing windows alphabetically. The name of the project window appears above them. Select a window's name to bring it to the front. The first nine editing windows also have command key equivalents to bring them to the front: Command-1 through Command-9. Command-0 (Command-Zero) brings the project window to the front. A modified file has a diamond (◊) in front of its name.

If you want to rearrange the editing windows so all their title bars are visible, choose the **Arrange** command in the **Windows** menu. It displays the dialog shown in Figure 10-5.

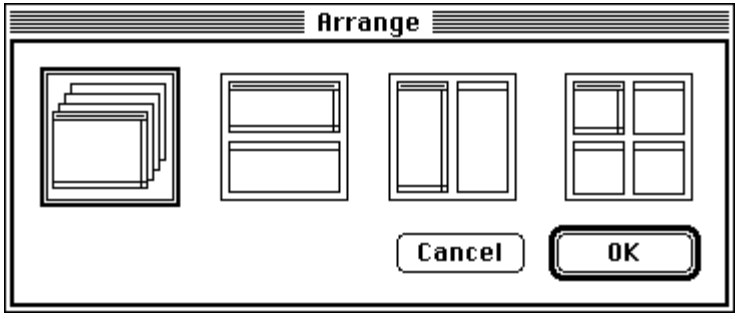

Figure 10-5 The Arrange dialog

The four pictures represent different ways to arrange your editing windows, shown in detail in Table 10-3. To rearrange your windows, double-click one of the pictures, or select one of the pictures and

click OK. To leave your windows arranged as they are, click Cancel.

Picture	Description
	Stack. Stagger the windows. Each window is slightly below and to the right of the window beneath it. Only the title bars of the windows in the back are visible.
	Vertical tile. The front window takes up the top half of the screen. The other windows are stacked in the bottom half.
	Horizontal tile. The front window takes up the left half of the screen. The other windows are stacked in the right half.
	Full tile. Tile the windows so they're all visible. The front window is in the top left corner of the screen.

Closing and Saving Files

It's a good idea to save your work every fifteen minutes or so, in case something horrible happens. Power failures and machine crashes can occur at the most inopportune times.

Closing a file

To close a file, click its Edit window's close box. If you've made changes and haven't saved the file, the editor asks if you want to save it before closing. You can also use the **Close** command from the **File** menu to close a file.

Saving a file

To save a file without closing it, use the **Save** command from the **File** menu. If you've never saved the file before (that is, if its edit window is untitled), a standard file dialog asks you to name the file.

Saving a file with a different name

The THINK editor gives you two ways of saving a file under a different name. The **Save As** command asks you to name a new file, then saves the contents of the edit window under that name.

If the file is part of your project (it's in your project window), its name changes there too.

The **Save a Copy As** is similar to the **Save As** command except that it doesn't change the name of the file. It saves the contents of the current edit window under a new name but lets you continue editing the original file under its original name. The project window remains unchanged. This is a good way to make backup copies without mistakenly editing the backup.

Saving and closing all open files

To save all the open files, use the **Save All** command in the **Windows** menu. Using this command is the same as using the **Save** command on each window.

You can close all the open files by choosing the **Close All** command from the **Windows** menu or by holding down the Option key as you click the close box of any editing window. If a file hasn't been saved, the editor asks you if you want to save it.

Saving files automatically

The THINK editor saves files for you automatically when you close them if you uncheck the Confirm saves check box in the Editor page of the **THINK Project Manager options** dialog.

Executing MPW Commands

If you have ToolServer and are using System 7, you can execute MPW commands in a THINK editor window. ToolServer is available separately from APDA and is a scaled-down version of MPW (Macintosh Programmer's Workshop). It executes commands that other applications send it with AppleEvents. MPW is Apple's

development environment that uses a command-line interface, much like Unix or MS-DOS.

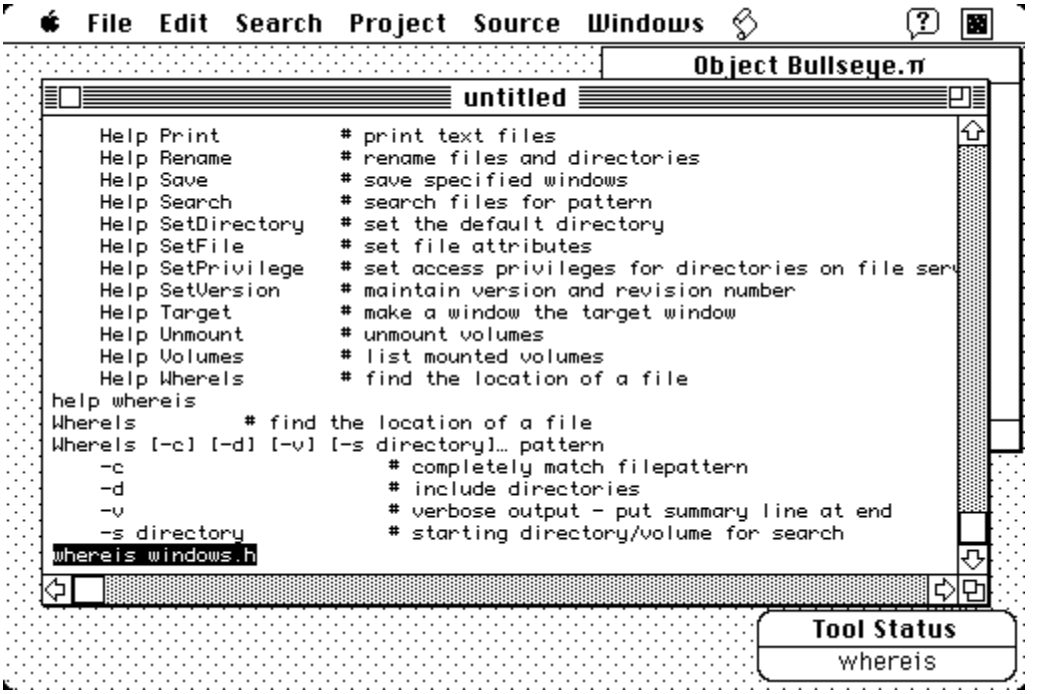

Figure 10-6 Executing an MPW command

To set up ToolServer, follow the installation instructions that come with it. Then make an alias of the ToolServer application, name it `ToolServer`, and place it in the `Tools` folder in your THINK Project Manager folder. If the alias isn't named `ToolServer`, the THINK editor isn't able to find it.

To execute an MPW command, type the command and press Option-Enter. (Option-Return doesn't work.) To execute several commands, select them and press Option-Enter. To re-execute a command, place the insertion point anywhere in its line and press Option-Enter. The THINK editor sends the text you selected or the text in the same line as the insertion point to ToolServer. ToolServer works in the background and displays its status in the Tool Status window. If it needs to bring up a dialog, such as a Commando dialog, ToolServer comes to the front. When ToolServer is finished, the THINK editor pastes its response under the insertion point. If the

window that you entered the command in isn't the frontmost window, the THINK editor beeps. If you closed the window, the THINK editor opens a new window to paste the response in.

Using Markers

The THINK editor recognizes the markers that MPW puts in files, and MPW recognizes the markers that the THINK editor puts in files.

Markers are like bookmarks in a book. They help you get to a certain point in your program quickly. You can place markers at the beginning of each routine, or anywhere else in your program.

Adding markers

To put a marker in a file, place the insertion point in the line you want to mark or select some text in that line. Choose the **Mark** command in the **Search** menu. You see the dialog in Figure 10-7.

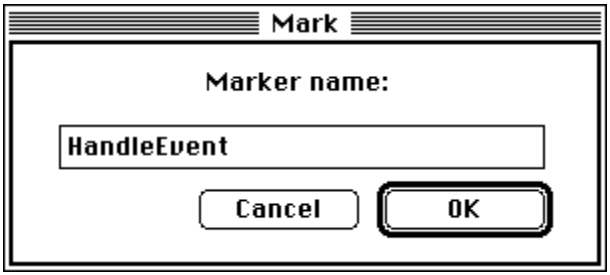

Figure 10-7 The Mark dialog

If you selected text, the editor uses that text as the default name. If you don't select text, the editor uses the word nearest the insertion point as the default name. Edit the default name, if you want, and click OK. If you don't want to insert a marker after all, click Cancel.

Note

Make sure that your marker has a unique name. The THINK editor doesn't check to see whether the name has already been used. If several markers have the same name, that name is listed several times in the marker pop-up menu.

Holding down the Option key while clicking a title bar brings up a menu of the file's header files, described in "Opening header files" earlier in this chapter .

Going to a marker

To go to a marker in a file, hold down the Command key and click the title bar of the file's editing window. A pop-up menu of the file's

markers is displayed, sorted alphabetically, as shown in Figure 10-7. Choose the marker you want to go to, and the editor scrolls to that marker. If you marked a text selection, the editor selects that text. If you marked an insertion point, the editor moves the insertion point there.

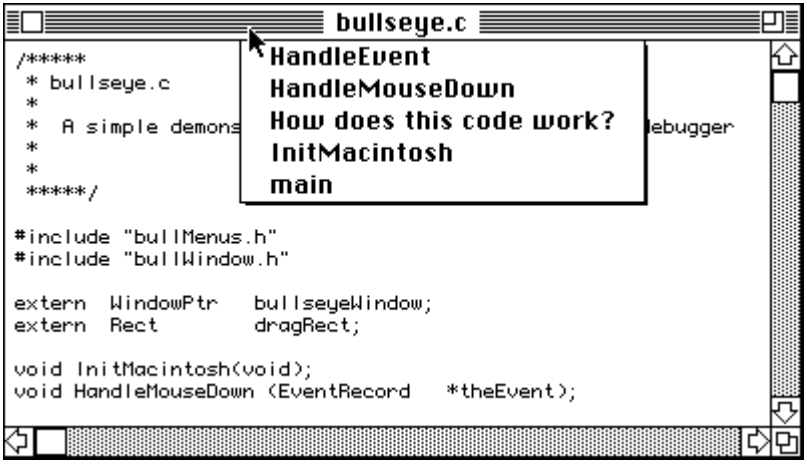

Figure 10-8 The marker pop-up menu

Deleting a marker

To delete a marker, choose the **Remove Marker** command from the **Search** menu. You see the dialog shown in Figure 10-7.

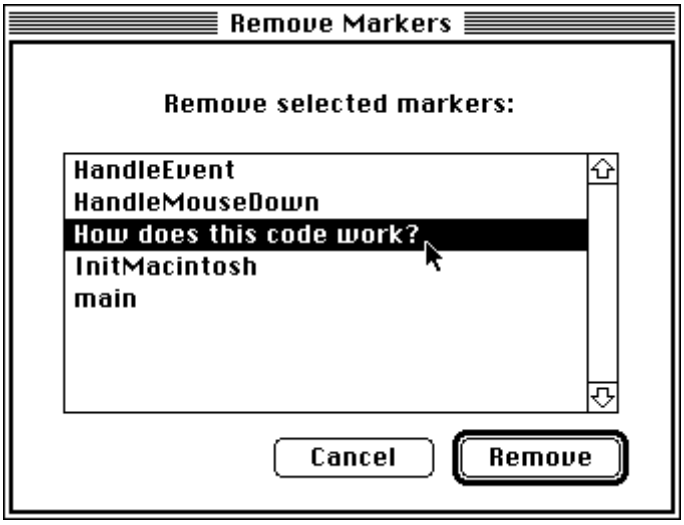

Figure 10-9 The Remove Marker dialog

Select the markers you want to delete and click OK. If you change your mind, click Cancel. The next time you see the marker pop-up menu, the deleted markers will be gone.

Using Your Own Editor

If you don't want to edit your files with the built-in THINK editor, you can use a text editor of your own choosing. How closely your editor works with the THINK Project Manager depends on whether your editor is THINK-compatible.

Using a THINK-compatible editor

If you're running System 7 and the editor is THINK-compatible, the editor and the THINK Project Manager can work together by using AppleEvents. When you double-click a file name in your project window, the editor opens the file. The THINK Project Manager knows automatically when you change a file and what its new contents are. The THINK Project Manager can direct the editor to perform searches and display errors.

Check your editor's documentation to see if it's THINK-compatible and learn how it works with the THINK Project Manager.

To use a THINK-compatible editor, follow these steps:

1. Create an alias to your editor, name the alias `Editor`, and put it in the `Tools` folder.

2. On the Editor page of the **THINK Project Manager options** dialog, select the Use external editor option.

Using another editor

If your editor is not THINK-compatible, you can still edit your source files with it, but it doesn't work together with the THINK Project Manager. Your editor can't open files when you double-click them in the project window, or work with the THINK Project Manager to perform searches and display errors.

Note

Before you run or build a project, you must save all the files that you've modified in the editor.

To use an editor that's not THINK-compatible, select the Always check file dates option in the Preferences page of the **THINK Project Manager options** dialog. When the THINK Project Manager runs or builds a project, it now checks the modification dates of the project's files and recompiles the files if their dates are later than those stored in the project for the files.

Searching and Replacing

The THINK editor offers a wide range of search and replace capabilities. You can:

- Find a string in a file
- Replace one string with another
- Find a string in any file in your project
- Find the definition of a symbol
- Find and replace strings that match a pattern (grep)

Finding a string

Use the **Find** command from the **Search** menu when you want to find a string. You see the dialog box in Figure 10-10.

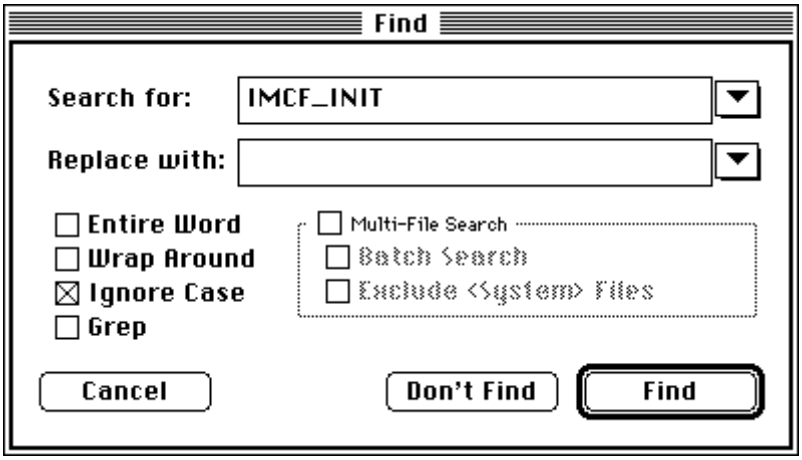

Figure 10-10 The Find dialog box

Type the string you're looking for in the Search for field. You can also use the pop-up menu to the right of the field to choose from the past five strings you entered in the field, as shown in Figure 10-10.

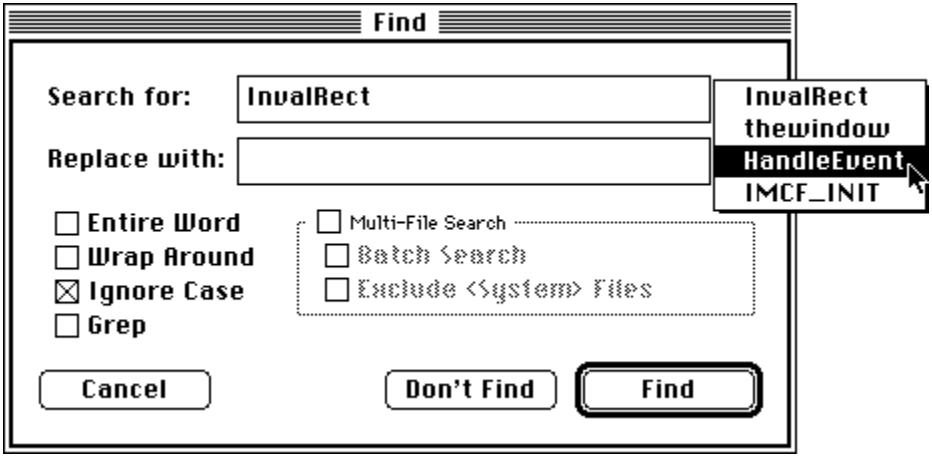

Figure 10-11 The Find dialog with a pop-up menu

When you click the Find button, the editor searches for the string in the file you're editing. If it finds the string, it scrolls to and highlights that string. If it can't find the string, it beeps.

Because the string you've found is highlighted, you can replace it just by typing in a replacement string.

To find the next instance of the string, use the **Find Again** command in the **Search** menu.

Search options

The four check boxes on the left side of the **Find** dialog let you specify how the editor looks for your string.

If you check the Entire Word option, the editor matches only whole words. This option is useful when you're looking for a one-letter variable name or a word that commonly appears inside other words, such as `handle`.

The editor usually searches from the insertion point to the end of the file. If you check the Wrap Around option, it searches the entire file for your string. The search begins from the insertion point (or the end of the selection). If the editor doesn't find your string by the time it reaches the end of the file, it flashes the menu bar and searches from the beginning to the insertion point.

When the Ignore Case option is checked, the editor matches the search string, regardless of case. If this option is off, the case of the strings must match exactly.

When the Grep option is on, you can search for patterns with a feature as powerful as the Grep utility on many Unix systems. For more information, see "Searching for a Pattern (Grep)" later in this chapter.

When the Multi-File Search option is on, you can search for a string in several files in your project. When the Batch Search option is on, the THINK editor compiles a list of all occurrences of your search string. You can browse this list at your leisure. For more information, see "Searching through multiple files" later in this chapter.

*For more information on the **Options** commands, see "Setting Project Options" in Chapter 8, "Organizing Projects."*

Note

You can set the defaults for these options on the Editor page of the **THINK Project Manager options** dialog, shown in Figure 10-12. When you set the options in the **Find** dialog, they apply only to the current session. When you set the options in the **THINK Project Manager options** dialog, they are the permanent defaults for the current project or for all new projects.

The "Batch Search" and "Exclude <System> Files" options are described later in this chapter in "Searching through multiple files" later in this chapter.

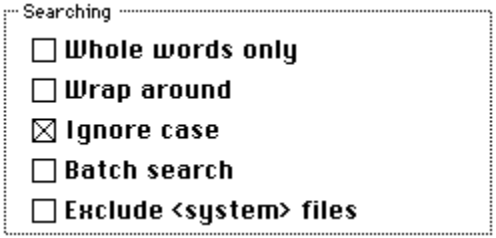

Figure 10-12 The Searching options

Replacing a string

If you want to replace some but not all instances of the search string, enter a replacement string in the Replace with field of the **Find** dialog box. You can also use the pop-up menu to the right of the field to choose from the past five strings you entered in the field.

When the editor finds the first occurrence, use one of these commands:

- **Find Again**, to go on to the next instance
- **Replace**, to replace it with the replacement string
- **Replace and Find Again**, to replace the current instance and immediately go on to the next one

Replace All replaces every instance of the search string in the file with the replacement string. If you don't type in a replacement string, it deletes every instance of the search string (that is, it replaces it with nothing).

Setting things up for searching later

In addition to the Find button ("Go ahead with the search") and the Cancel button ("Pretend I never invoked this command"), there is a

Don't Find button. Clicking this button sets up the search and replace strings and the option settings without doing the search. To go ahead with the search later, use the **Find Again** command.

You might use the Don't Find button when you realize that the insertion point is not where it should be to start the search. Click the button, move the insertion point to the proper place, then use **Find Again** to find your string.

Note

The **Find Again** command looks for the string you've entered in the search string.

Finding nonprinting characters

To look for tab and return characters, hold down the Command key as you type these characters into the Search for and Replace with fields. To insert other nonprinting characters, use the **Copy** command to copy them into the Clipboard, then **Paste** them into the Search for and Replace with fields.

A Return signifies the beginning or end of a line in a string search.

Searching through multiple files

The Multi-File Search option lets you look for a string in more than one file in your project. This feature is useful when you're tracking down undefined or multiply defined symbols, or when you're changing the number of parameters to a function and need to all the references to it.

To look for a string in more than one file, check the Multi-File Search check box in the **Find** dialog box.

A dialog box displays all text files associated with the project.

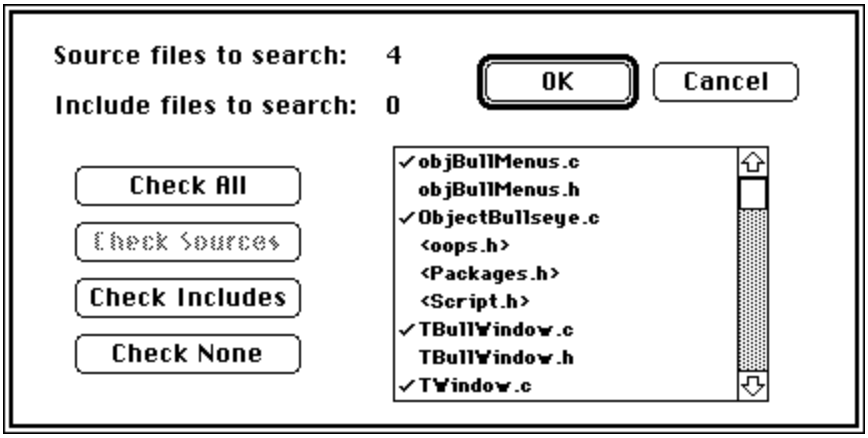

Figure 10-13 The Multi-File Search dialog

Scroll through the list and click individual files to select them. A small check mark appears next to the file name. If a file is already selected, clicking its name removes the check mark. You can use the buttons in the dialog box to Check All, Check None, Check Sources, or Check Includes files. You can also use these Command-keys instead of the button:

Button	Command-key equivalent
Check All	Command-A
Check Sources	Command-S
Check Includes	Command-H
Check None	Command-N

Table 10-3 Command keys in the Find dialog

When you've checked the files you want to search, click OK to return to the **Find** dialog box. If you selected any files, the Multi-File

Search option is checked and two more options appear under it, as shown in Figure 10-14.

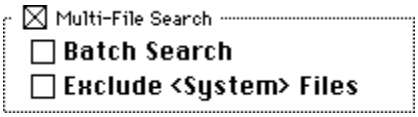

Figure 10-14 The Multi-File Search options

If you check the Batch Search option, the THINK editor writes out all the matches to the Search Results window and you view them after it's finished searching. Otherwise, the editor displays each match in an editing window as it's found.

If you check the Exclude <System> Files option, the THINK editor does not search files in the THINK tree, even if you selected them. This option is especially useful when you're working on a THINK Class Library project and you want to search only the files you created.

Note

You can set the defaults for these options on the Editor page of the **THINK Project Manager options** dialog. When you set the options in the **Find** dialog, they apply only to the current session. When you set the options in the **THINK Project Manager options** dialog, they are the permanent defaults for the current project or for all new projects.

*For more information on the **Options** commands, see "Setting Project Options", Chapter 8, "Organizing Projects.*

When you click Find, the THINK editor looks for the search string in the checked files, starting with the first one checked. If the Batch Search option is on, it writes out the file name, line number, and contents of the line for each match to the Search Results window, as shown in Figure 10-15. When it's finished searching, it beeps and selects the first match in the window. To look at that first match, press Return. The THINK editor opens the file in a window, moves to the line that contains the match, and selects the search string. To go to the next match, select **Go to Next Match** in the **Search** menu.

To go to the previous match, select **Go to Previous Match**. To view any match in the list, select it and press Return, or double-click it.

Figure 10-15 The Search Results window

Note

If you close the project while the Search Results window is still open, the THINK Project Manager saves the window's contents and restores them when you open the project again.

If the Batch Search option is off, the THINK editor shows you each match as it's found. The THINK editor opens the file and selects the search string. At this point, you can edit the file or, if you want to search further in the current file, you can use the **Find**, **Find Again**, **Replace**, and **Replace All** commands to work within the current file. When you're ready to go on to the next file, use the **Find in Next File** command.

Note

When the Batch Search option is off, multi-file search finds the first instance of the search string in each file. To find additional instances, use the **Find Again** command. Once you issue a **Find in Next File** command, the THINK editor looks in the next file you checked, even if there are additional instances of the search string in the current file.

Here's another example of when you would want to use the Don't Find button. Suppose that in a multi-file search with Batch Search off, you decide you really want to set the Match Words option. Open the **Find** dialog and change the option. If you were to click the Find

button, the search would go on to the next file. So instead you click the Don't Find button, and continue using the **Find Again** and **Find in Next File** commands.

Disabling multi-file search

Entering a new search string cancels a multi-file search. To enter a new search string without cancelling a multi-file search, bring up the **Find** dialog. Click the Multi-File Search check box. A list of all text files appears. Click OK to accept all the checked files, then enter the search string in the Search for field.

To cancel a multi-file search without going through the multi-file selection dialog, hold down the Option or Command key as you click the Multi-File Search check box.

Searching for a Pattern (Grep)

In addition to the search and replace functions described in the previous section, the THINK editor also provides a powerful pattern search capability called Grep. The Grep search option in the THINK editor is based on the Grep utility on Unix systems. If you're familiar with this kind of pattern matching, this section should present no problems. If pattern searching is new to you, experiment with this feature before you using it on a real file.

Warning

The editor looks for patterns only when the Grep option is on.

Patterns

A pattern is a description of a set of strings rather than a specific string. For example, you can build a pattern that means "any word that begins with P." Or a pattern that means "any function call with `&event` as an argument."

Patterns can't span lines. This means that you can't write a pattern that means "three consecutive lines that begin with a, b, and c."

Simple patterns

The simplest patterns match a single character.

- Any character, with the following exceptions noted, is a pattern that matches itself.

For example, the pattern `2` matches a character `2`. If you've checked Ignore Case in the **Find** dialog box, any letter matches both its uppercase and lowercase equivalent. So either `a` or `A` matches both `a` and `A`.

- The character `.` is a pattern that matches any character.
- The character `\` followed by any character except `( )`, `<`, `>`, or one of the digits `1` through `9` is a pattern that matches that character.

For example, `\.` matches `.` and `\\` matches `\`.

- A string of characters *s* surrounded by `[` and `]` is a pattern `[`*s*`]` that matches any one of the characters in the string `s`. The pattern `[^`*s*`]` matches any character not in the string *s*. If *s* is a string of three characters in the form `a-z`, this represents all of the characters from `a` to `z` inclusive. All other characters in *s* are taken literally. To include the character `]` in *s*, you must use it as the first character. To include the character `-` in *s*, you must use it at the beginning or end of *s*.

For example, the pattern `[A-Za-z0-9]` matches any alphanumeric character. The pattern `[^!-~]` matches any nonprinting ASCII character. The Ignore Case option has no effect on anything between brackets.

Complex Patterns

To match strings, not just individual characters, you need patterns that match consecutive sequences of characters. One way of doing this is to append a `*` to the end of one of the simple patterns.

- A pattern *x* followed by a `*` is a pattern *x*`*` that matches zero or more consecutive occurrences of characters matched by *x*.

For example, the pattern `@*` matches a string containing any number of at-signs. If the string does not begin with an at-sign, or if it contains no at-signs at all, then the pattern matches the empty string at the beginning of the string to be matched. Later you will see why this is useful.

You can put patterns together to form more complex patterns:

- A pattern *x* followed by a pattern *y* forms a pattern *xy* that matches any string *ab*, where *a* matches *x* and *b* matches *y*.

For example, the pattern `P.` matches any two-letter string consisting of P and any other character.

- You can concatenate the compound pattern *xy* with another pattern *z*, forming the pattern *xyz*.

To put all this together, consider the pattern `(.*)`. This pattern matches any string enclosed in parentheses. This includes the string `()`, since the sub-pattern `.*` matches the empty string between the `(` and the `)`.

Does this pattern match the string `(())`? Because the sub-pattern `.*` matches any number of occurrences of all characters, it might appear that the pattern will match just the `(()` and not the very last `)`? That isn't the case because any sub-pattern of the form *x*`*` in a pattern *x*`*`*y* matches the largest number of occurrences of whatever *x* matches that still allows a match to *y*. In matching `(())` against the pattern `(.*)`, only the inner pair of parentheses matches the sub-pattern `.*`, so the pattern matches `(())`.

Sub-patterns

Grep has a way of remembering sub-patterns so you can use them again as part of even more complex patterns. This becomes somewhat complicated.

- A pattern surrounded by `\(` and `\)` matches whatever the sub-pattern matches.

For example, `\(a[b-y]z\)` matches the same thing as `a[b-y]z`.

- A `\` followed by *n*, where *n* is one of the digits `1` through `9`, matches whatever the *n*th `\(` `\)` sub-pattern matched. You can add a `*` to a `\`*n* pattern to form a pattern `\`*n*`*` that matches zero or more occurrences of whatever `\`*n* matched.

For example, to find two repeated words (like "the the"), you might use a pattern such as this: `\([a-z][a-z]*\) \1`. This pattern matches a space, any sequence of letters, a space, and the same sequence of letters. Note that `\1` is not

a reapplication of the pattern. Instead it becomes whatever the first `\( \)` pair matched.

Constraining patterns

Finally, you can constrain patterns to match only if they meet certain conditions in the context outside the string.

- A pattern surrounded by `\<` and `\>` matches whatever the pattern matches, provided that the first and last characters of the matched string match `[A–Za–z0–9_]` and that the characters immediately surrounding the matched string don't match `[A-Za–z0–9_]`. In other words, the pattern matches only if the string begins and ends on a word boundary. If you've checked Match Words in the **Find** dialog box, the entire pattern that you enter is treated as though it were surrounded by `\<` and `\>`.

For example, to find occurrences of repeated words, even if they're not surrounded by spaces, you would use the pattern `\(\<[a-z][a-z]*\>\)[^a-z]*\1`.

- A pattern *x* preceded by a `^` forms a pattern `^`*x*`.` If `^`*x* is not preceded by any other pattern, it matches whatever *x* matches as long as the first character *x* matches occurs at the beginning of a line.

- A pattern *x* followed by a `$` forms a pattern *x*`$`. If the pattern *x*`$` is not followed by any other pattern, it matches whatever *x* matches as long as the last character that *x* matches is at the end of a line. If the pattern *x*`$` is followed by another pattern, then the `$` is taken literally.

These last two items constrain pattern matches to begin or end at line boundaries, and they can be combined to constrain a pattern to match an entire line only.

Replacing with Grep

You can use Grep not only to search for strings, but also to replace them. The following special characters let you alter the replacement string.

- Each occurrence of the character & is replaced with whatever the entire pattern matched.

For example, if you wanted to add a `P` to the beginning of every word that ended with `ptr`, you would search for `\<.*ptr\>` and replace it with `P&`.

- Each occurrence of `\n`, where *n* is one of the digits `1`through `9`, is replaced by whatever the *n*th occurrence of `\n` matched.

For example, to change all strings such as`#define FOO 1` to `FOO = 1`, search for

```
#define \(\<[A-Za-z0-9][A-Za-z0-9]*\>\)
\(\<.*\>\)
```

and replace it with

```
\1 = \2
```

- Each occurrence of a string `\x`, where *x* is not one of the digits `1`through `9`, is replaced by *x*.

Grep Examples

Grep is not easy to learn. Following are some typical examples to give you some help.

Suppose that you've written a Macintosh application and you've forgotten to put a `\p` at the beginning of your strings to signal to the compiler to make them Pascal strings rather than C strings. You can change all your C strings to Pascal strings by specifying

```
"\([^"]*\)"
```

as the search pattern and

```
"\\p\1"
```

as the replacement string.

To convert

```
sym equ (expr+4)     ; a comment
```

to

```
#define sym (expr+4) /* ; a comment */
```

search for

```
\(\<.*\>\)[space tab]*\<equ\>\([^;]*\)\(.*\)
```

and replace with

```
#define \1 \2 /* \3 */
```

Explanation:

- `\<.*\>` matches a symbol.
- The surrounding `\(` and `\)` lets you use the symbol in the replacement string as `\1`.
- The `[space tab]*` matches any number of spaces or tabs between the symbol and the key word `equ`. (The words *space* and *tab* stand for the characters here, because you can't see them on paper. To enter a Tab, press Command-Tab in the dialog box.)
- `\<equ\>` matches the word `equ`. It does not match `equ` if it is part of another word, for example, `equal`. `\<equ\>` is not surrounded by `\(` and `\)` because it is thrown away in the replacement string.
- `[^;]*` matches an expression formed by any number of characters up to but not including a ; (semi-colon).
- The surrounding `\(` and `\)` lets you use the expression in the replacement string as `\2`.
- The `.*` matches the comment that is the rest of the line.
- The surrounding `\(` and `\)` stores the comment as `\3`.
- If there is no ; (semicolon) in the line, then `\2` consists of everything after the `equ` to the end of the line, and `\3` is an empty string.

To convert `$HHHH` to `0xHHHH`, where `H` is a hexadecimal digit, search for

```
$\([0-9A-Fa-f][0-9A-Fa-f]*\)
```

and replace with

```
0x\1
```

Explanation:

- `$` matches a `$`. `[0-9A-Fa-f]` matches one hex digit.
- `[0-9A-Fa-f][0-9A-Fa-f]*` matches one or more hex digits. (The pattern `[0-9A-Fa-f]*` matches zero or more hex digits.)
- The surrounding `\(` and `\)` lets you remember the hex digits in the replacement string as `\1`.

Save complicated Grep search and replace strings in a file, so you can copy and paste them into the **Find** dialog box.